REPRESENTATION AND REASONING MODELS FOR C3 ARCHITECTURE DESCRIPTION LANGUAGE

Abdelkrim Amirat, Mourad Oussalah

REPRESENTATION AND REASONING MODELS FOR C3 ARCHITECTURE DESCRIPTION LANGUAGE

Abdelkrim Amirat and Mourad Oussalah
Laboratoire LINA CNRS UMR 6241, Université de Nantes
2, Rue de la Houssinière, BP 92208,
44322 Nantes Cedex 03, France
{abdelkrim.amirat, mourad.oussalah}@Univ-Nantes.fr

Keywords: Model, Representation, Hierarchy, Connector, Architecture.

Abstract: Component-based development is a proven approach to manage the complexity of software and its need for customization. At an architectural level, one describes the principal system components and their pathways of interaction. So, Architecture is considered to be the driving aspect of the development process; it allows specifying which aspects and models in each level needed according to the software architecture design. Early Architecture description languages (ADLs), nearly exclusive, focus on structural abstraction hierarchy ignoring behavioural description hierarchy, conceptual hierarchy, and metamodeling hierarchy. In this paper we focus on those four hierarchies which represent views to appropriately "reason about" software architectures described using our C3 metamodel which is a minimal and complete architecture description language. In this paper we provide a set of mechanisms to deal with different levels of each hierarchy, also we introduce our proper structural definition for connector's elements deployed in C3 Architectures.

1 INTRODUCTION

Nowadays, there is a completely new approach to building more reliable software systems which consist to decompose large and complex systems into smaller and well-defined units called software *components*. A component-based application is a collection of individual components, which are interconnected via well-defined *connectors* between their interfaces.

Component that have no externally observable internal structure, while having real implementation in certain programming language, are called *primitive components*. Components containing nested subcomponents, i.e. components with observable internal structure, are called *configurations*. So, components, connectors and configurations commonly referred to as *core elements*, are typically defined in an Architecture Description Language (ADL) (Allen, 1997).

Software architecture researchers need extensible, flexible architecture descriptions languages and equally clear and flexible mechanisms to manipulate these core elements at the architecture level.

There is not today, nor has there ever been, a clear consensus on a definition of software architecture. Recently Medvidovic (Medvidovic, 2007) gives the following definition for software architecture "*A software system's architecture is the set of design decisions about the system"*. So, if those decisions are made incorrectly, they may cause your project to be cancelled. However, these design decisions encompass every aspect of the system under development, including: design decisions related to system structure, design decisions related to behaviour also referred to as functional, design decisions related to the system's non functional properties. Also, we can elicit other design decisions related to the development process or the business position (*product-line*).

The "first-generation" ADLs all shared certain traits. They all modeled the structural and, with the exception of Acme (Garlan, 2000), functional characteristics of software systems. They invariably took a single, limited perspective on software architecture. In this paper we introduce further perspectives which are complementary to the structural one. The majority of ADLs proposes, like reasoning model, only sub-typing as a mechanism for specialization (e.g. Acme, C2). Otherwise, for

the rest of ADLs, they propose their own ad hoc mechanisms based on algorithms and methods designed specially for them. Based on a bread survey of architecture description notations and approaches, we identified that ADLs capture aspects of software design centred around a system's *Component*, *connectors*, and *configurations*. The core elements of our model are basically defined around these tree elements. So, from this we derive the name of our model **C3** for **C**omponent, **C**onnector, and **C**onfiguration. The rest of the paper is organized as fellows. In section 2 presents our research motivations. Section 3 describes the C3 metamodel. The last section presents a conclusion about this work.

2 MOTIVATION

Our motivation in this work is to develop a generic model for the description of software architectures which must be minimal and complete. It is minimal because we are only interested by the core concepts in each ADL. And complete because with this minimum of concepts the architect will be able to describe any required structure he needs to realize using those concepts and a set of predefined mechanisms. However, describing only the architecture structure is not sufficient to provide correct and reliable software systems. In this paper we are going to focus more on representation architecture model and to reason about its elements following four different types of hierarchies. Each of those hierarchies provides a particular view on the architecture. We expect from our approach to provide more explicit and better clarified software architecture. Mainly the approach is developed to:

- Make explicit the possible types of hierarchies being used as support to reason about the architecture, with the different possible levels in each hierarchy.
- Show semantics conveyed by every type of hierarchy by providing the necessary mechanisms used to connect elements of in the same hierarchical level and the mechanisms used to connect elements of every level with the elements of the adjacent levels.
- Allow introducing various mechanisms of reasoning within the same architecture according to the requirements of the system in a specific application domain.
- Establish the position of existing mechanisms developed for reasoning with regard to our referential.

3 THE C3 METAMODEL

In order to have a complete C3 metamodel, we have defined mainly two complementary models to describe and reason about system's architecture. We use a *representation model* to describe architectures based on C3 elements and we use a *reasoning model* to understand and analyse the representation model.

3.1 Representation Model

The core elements of the C3 representation model are components, connectors, and configurations, each of these elements have an interface to interact with its environment like depicted in Figure 1.

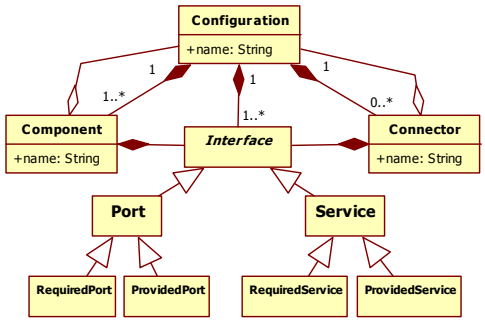

Figure 1: Basic elements of C3 metamodel

Components represent the primary computational elements and data stores of a software system. Intuitively, they correspond to the boxes in box-and-line descriptions of software architecture. In C3, each component can have one or more ports. Ports are the interaction points between components and their environments.

Connectors are very important entities that unfortunately are not dealt with by all conventional component-based models. In C3, connectors represent interconnections among components to support their interaction and they are defined explicitly and considered as first class entities by separating their interfaces from their implementation. Figure 2 illustrates our contribution at this level which consists in enhancing the structure of a connector by encapsulating the attachment links inside its definition. So, the application builder will have to spend no effort in connecting connectors with their compatible components or configurations. Consequently, the task of the developer consists only to select the suitable type of connector which is compatible with the types of components and/or

configurations which are expected to be connected by it this last one (Amirat, 2007).

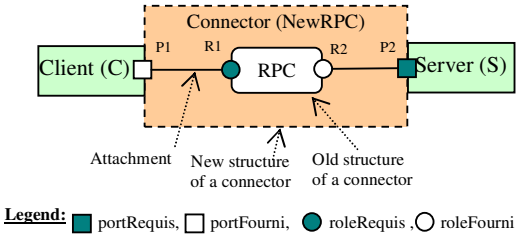

Figure 2: The new structure of a connector

So, by encapsulating attachments links inside the connector, we can give the following connector definition.

Connector_NewRPC (C.P1 , S.P2) {
Roles R1, R2;
Services sendRequest , receiveRequest;
Properties {*List of properties*}
Constraints maxClient = 3 ;
Glue R1=R2 // simple mapping function
Connexions {RPC.R1 to C.P1, RPC.R2 to S. P2} }

Configuration. In C3 each component or connector is perceived and handled from the outside as a primitive element. But their inside can be real primitive elements or composite with a configuration encapsulating all the internal elements. So, in C3 we define configurations as first-class entities. They represent graphs of components and connectors to describe how they are interconnected to each other. A configuration may have an interface specified by a set of ports and services. Each port is perceived like a bridge between the internal environment of the configuration and the external one. So, the different elements of the architecture are connected through their interfaces. Thus the interface types of are checked if they are compatible or not (*interface matching*). Consequently, the consistency of elements assembly is controlled syntactically.

3.2 Reasoning Model

In our approach we plan to analyze the software architecture by using different hierarchy views where each hierarchy is investigated at different levels of representations. C3 reasoning model is defined by four hierarchies. Each hierarchy represents a specific view on the C3 representation model different from the others.

Those four hierarchies are: 1- The structural hierarchy used to explicit the different nested levels of abstraction that the system's architecture can have. 2- The behavioural hierarchy to describe the different levels of system's behaviour hierarchy generally represented by protocols. 3- The conceptual hierarchy to provide the libraries of element types corresponding to structural or behavioural element at each level of the architecture description. 4- The metamodeling hierarchy to locate from where our metamodel is coming and what we can do with it. Obviously those two sides will belong to the pyramid of abstraction hierarchies defined by Object Management Group (OMG, 2007).

3.2.1 Structural Hierarchy (SH)

Structural hierarchy has to provide the structure of particular system architecture in terms of the architectural elements. The majority of academic ADLs like Aesop, MetaH, Rapide, SADL, and others (Matevska-Mayer, 2004) or the industrial ones like CCM, EJB or .Net (Pinto, 2005) allow only a flat description of software architectures. Using those ADLs architecture is described only in terms of components connected by connectors without any nested elements. This design choice was made in order to simplify the structure and also by lack of concepts and mechanisms that respectively define and manipulate configurations of components and connectors.

In our metamodel the structure of an architecture is described using components, connectors, and configurations, where configurations are composite elements. Each element in this configuration can be a primitive (with a basic behaviour scenario) or a new configuration which contains another set of components and connectors, which in their turn can be primitive or composite material, and so on.

However, C3 allows the representation of architecture with the necessary number of abstraction levels (L_n, L_{n-1}… L_0), where n depends on the complexity of the system. Practically all architectural solutions for domain problems have a nested hierarchical nature. Thus, real software architecture can be viewed as a graph where each internal node of this graph represents a configuration and each leaf-node represents a primitive component arcs between nodes represent connectors.

As a simple illustrative example, Figure 3 depicts the client-server (CS) architecture. We analyse this example from the provided view of each type of hierarchy introduced in this paper. In the following figures we use numbers to represent architecture elements.

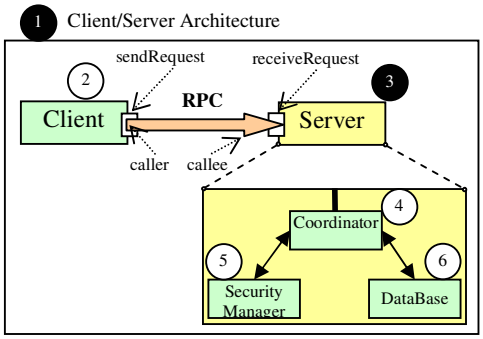

Figure 3: CS architecture

To describe the structural hierarchy we use the following three types of connectors.

Composition/Decomposition Connector (CDC) used to link each configuration to its underling elements. Therefore, this type of connector allows the propagation of information among elements of the structural hierarchy. And we can determine the child's or the configuration, if it is the case, of each element deployed in the architecture. Figure 4 illustrates how to use CDC connector to represent composition/decomposition relationships in the client-server example.

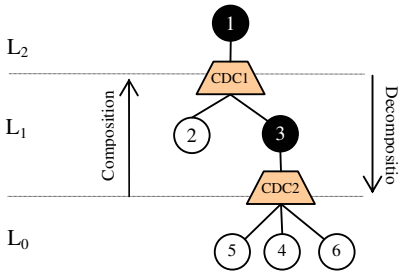

Figure 4: Structural hierarchy with CDC connector

Attachment Connector (AC) we use this type of connector to establish service-connections between architectural elements[1] deployed in the same level of abstraction as illustrated in Figure 5. In some ADLs this type of connector in called *assembly connector* and represented by first class entity e.g. Acme (Garlan, 2000). Inside the AC connector the glue code specifies the interaction protocol among communicating elements.

[1] From now on, we use the term architectural element to mean a component or a configuration.

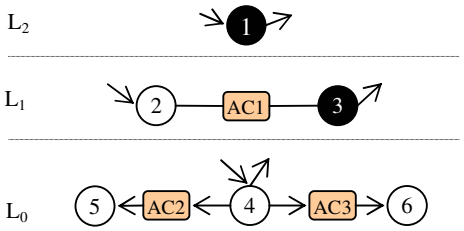

Figure 5: Structural Hierarchy with AC connectors

The description of the RPC connector (AC1) used to connect the client (node 2) to the server (node 3) is the following:

Component Client {Port {send-request}**}**
Configuration Server {Port {receive-request}**}**
Connector AC1 (Client, Server) **{**
Roles {Caller, Callee}
Services {List of services}
Properties {maxRoles: integer = 2, Synchronous: Boolean = true}
Constraints {List of constraints}
Glue {caller = callee}
Connections {Client.send-request to RPC.caller
Server.receive-request to RPC.callee} **}**

At each level, in the structural hierarchy, we use a different set of mechanisms to deal with the input interfaces and the output interfaces. For this reason inputs are generally expended when we shift from (L_i) to (L_{i-1}) and outputs are compressed when we shift from (L_{i-1}) to (L_i). The data format will change when we change the level. So, it is necessary that mechanisms used at each level are not the same. From this observation we will define, in the following, a connector for expansion of inputs and compression of outputs.

Expansion-Compression Connector (ECC) we use this type of connectors to establish service-connections between configurations and their underling elements (Figure 6). In some ADLs this type of link is called *binding* like in Acme or *delegation* like in UML but they don't define it as a first class entity.

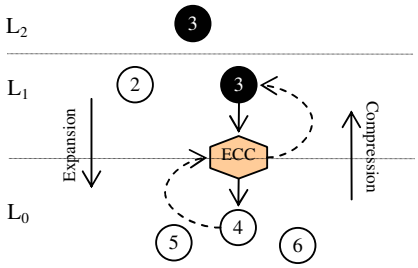

Figure 6: Structural hierarchy with one ECC connector

3.2.2 Behavioural hierarchy (BH)

The BH represents the description of the system's behaviour at different hierarchical levels. Each primitive element of the architecture has its own behaviour. The behaviour description associated with the highest level of the hierarchy represents the overall behaviour of the architecture (Lanoix, 2007). This behaviour is described by a global protocol P_1. The system architecture at this level is perceived as a black box with a set of required services and provided services. At lower level each component and configuration has its own protocol to describe its functionality. So, a protocol is used to specify the behaviour function of an architectural element by defining the relationship among the possible states of this element and its ability to produce coherent results. Figure 7 sketches how to decompose the client-server protocol P_1 at level L_2 into its sub-protocols at level L_1. This decomposition process produces two other protocols (P_2, P_3). By the same process P_3 protocol is decomposed to produce an other set of sub-protocols at the last level. The protocol leaves represent a primitive function of elements which are available in the library.

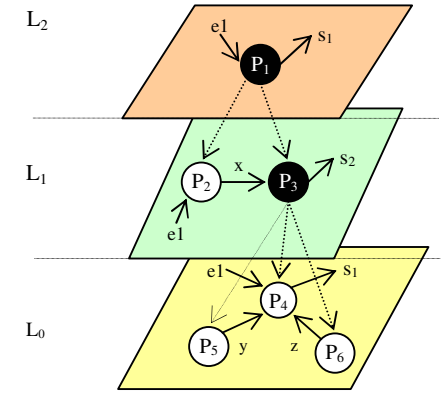

Figure 7: Plane representation of behavioural hierarchy

To explicit the different relationships among elements of the behavioural hierarchy we use the same set of connectors defined in the previous structural hierarchy, namely the CDC connector to compose and decompose behaviour elements, AC connector to link behaviours belonging to the same hierarchical level and ECC to expanse and compress exchanged information between behaviours.

3.2.3 Conceptual hierarchy (CH)

Through the mechanism of specialization the architect can create and classify element libraries according to architecture development needs in each target domain. The number of sub-type levels is unlimited. But we must remain at reasonable levels of specialization in order to keep compromise between the use and the reuse of the architectural elements. Those libraries represent the conceptual hierarchy (Frakes, 2005). To implement the conceptual hierarchy we use the following connector.

Specialisation/Generalisation Connector (SGC) is used to connect each element type to its super-type in the same level of abstraction. The conceptual hierarchy depicted in Figure 8 illustrates how to use the SGC connector to generate the five meta-type connectors from the first meta-connector defined by C3. The SGC used at this level is the bootstrap for the others meta-type connectors. Of course and by the same way we can use the SGC connector to specialise any architecture element.

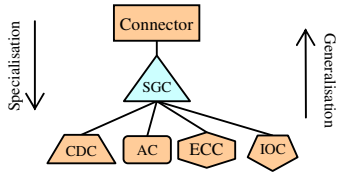

Figure 8: Conceptual hierarchy with SGC connector

3.2.4 Metamodeling hierarchy (MH)

The metamodeling hierarchy is defined by 4 abstraction levels (A_0,…,A_3). Each level (A_i) must conform to the description given above in $A_{(i+1)}$ level. The level A_3 conforms to itself. Symmetrically, each level (A_i) describes the inferior level $A_{(i-1)}$. A_0 is the application level (OMG, 2007).

Application level (A_0) is an instance of the architecture model (level A_1). At this level the developer has the possibility to select and instantiate elements any times as he needs to describe his application. Instances are created from element types defined at A_1 level. Elements are created and assembled with respect to the different constraints defined at A_1 Level.

Architecture level (A_1) at this level architecture is described using language constructions defined at A_2 level (e.g. C3 metamodel, UML 2.0). Thus, each

architecture model is an instance of the metamodel defined in the above level in the metamodeling hierarchy.

Meta-architecture level (A_2) defines the language or the notation used to describe architectures at A_1 level. Meta-architecture is also used to modify or adapt the description language. All operations undertaken at this level are always in conformance with the top level of the pyramid.

Meta meta-architecture (A_3) describes concepts and elements used to define any new architecture description language or new notation. In previous work we have defined meta meta-architecture model called MADL[2] (Smeda, 2005). So, C3 metamodel is defined in conformance with MADL. To connect each architectural element instance to its type at the above level we define the following connector:

Instance-Of Connector (IOC) is used to establish connection between element instances and their classifier defined in the above level. Figure 9 illustrates the connections between all components instances used at the client-server application (A_0) and their component types at the architecture level (A_1) and the connections between all component types at (A_1) level with the C3 meta-component. Those connections are realised using *Instance-Of* connectors (IOC).

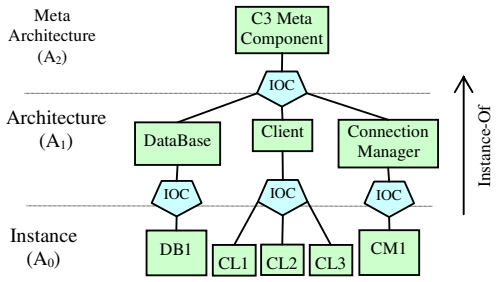

Figure 9: Internal view of IOC connector

4 CONCLUSION

In this work we have defined metamodel called C3 to describe software architecture and to reason about from different perspective views. The core elements of C3 are components, connectors and configurations. Elements are assembled using their interfaces. Syntactic and semantic compatibility are carried out using respectively interfaces-matching and protocols-matching. In this metamodel we mainly use structural hierarchy to describe the structural decomposition of the system, behaviour hierarchy to describe the behaviour decomposition, conceptual hierarchy to describe elements type libraries. Finally, we use the metamodeling hierarchy to show how we can modify the metamodel C3 and how to use it. Each hierarchy is supported and tooled by explicit mechanisms to provide the different form of connections. Contrary to the usual ADLs, which define only the attachment connectors, in C3 we define five types of connectors to deal with different connection forms. Structural and behavioural hierarchies use CDC, AC, and ECC connectors. Conceptual hierarchy uses SGC connector while metamodeling hierarchy uses the IOC connector.

[2] Meta Architecture Description Language

REFERENCES

Allen, R.J., 1997. A Formal Approach to Software Architecture. *PhD Thesis*. School of Computer Science, Carnegie Mellon University.

Amirat, A., Oussalah, M., Khammaci, T., 2007. Towards an Approach for Building Reliable Architectures. *In Proceeding of IEEE IRI'07*. Las Vegas, Nevada, USA, pp. 467-472.

Frakes, W. B., Kang, K., 2005. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*. vol.31 n.7, pp.529-536.

Garlan, D., Monroe, R.T., Wile, D., 2000. Acme: Architectural Description Component-Based Systems, Foundations of Component-Based Systems. Cambridge University Press, pp. 47-68.

Lanoix, A., Hatebur, D., Heisel, M., Souquières, J., 2007. Enhancing Dependability of Component-Based Systems. *Ada-Europe'07*, pp. 41-54.

Matevska-Meyer, J., Hasselbring, W., Reussner, R., 2004. Software architecture description supporting component deployment and system runtime reconfiguration. *WCOP'04*, Oslo.

Medvidovic, N., Dashofy, E., Taylor, R.N., 2007. Moving Architectural Description from Under the Technology Lamppost. *Information and Software Technology*. pp. 12-31. Vol. 49, No. 1.

OMG, 2007. Unified Modeling Language: Infrastructure. *from http://www.omg.org/docs/formal/07-02-06.pdf.*

Pinto, M., Fluentes, L., Troya, M., 2005. A Dynamic Component and Aspect-Oriented Platform. *The Computer Journal.* Vol.48 No. 4, pp. 401-420.

Smeda, A., Oussalah, M., Khammaci, T., 2005. MADL: Meta Architecture Description Language. *3rd ACIS International Conference SERA'05.* Pleasant, Michigan, USA, pp.152-159.